# Cerinţa; Bipartit complet

Se consideră două mulţimi nevide *A* şi *B*, cu proprietatea că formează o partiție a mulțimii *{1,2,...,n}.* Să se construiască un graf bipartit complet cu *n* vârfuri, bipartit peste partiţia formată din mulțimile *A* și *B*.

#include<fstream>

using namespace std;

ifstream fin("bipartitcomplet.in");

ofstream fout ("bipartitcomplet.out");

int a[101][101],n,x[101],y[101],viz[101],ok,m;

int main()

{

int i,j,k,u,v,p;

fin>>n>>k;

for(i=1; i<=k; i++)

{

fin>>x[i];

viz[x[i]]=1;

}

p=0;

for(i=1; i<=n; i++)

if(viz[i]==0)

{

p++;

y[p]=i;

}

for(i=1; i<=k; i++)

for(j=1; j<=p; j++)

{

u=x[i];

v=y[j];

a[u][v]=a[v][u]=1;

}

for(i=1; i<=n; i++)

{

for(j=1; j<=n; j++)

fout<<a[i][j]<<" ";

fout<<endl;

}

return 0;

}

# Cerinţa: Bipartit

Se dă lista muchiilor unui graf neorientat cu *n* vârfuri, etichetate de la *1* la *n*, precum si o mulțime *A* de vârfuri ale grafului. Considerăm mulțimea *B* formată din vărfurile grafului care nu aparțin lui *A*. Să se verifice dacă graful este bipartit peste partiția formată din mulțimile *A* și *B*.

#include<fstream>

using namespace std;

ifstream fin("bipartit.in");

ofstream fout ("bipartit.out");

int a[101][101],n,viz[101],ok,m;

int main()

{

int i,j,x,y,k;

fin>>n>>m;

for(i=1; i<=m; i++)

{

fin>>x>>y;

a[x][y]=a[y][x]=1;

}

fin>>k;

for(i=1; i<=k; i++)

{

fin>>x;

viz[x]=1;

}

ok=1;

for(i=1; i<=n-1 && ok==1; i++)

for(j=i+1; j<=n; j++)

if(a[i][j]==1 && viz[i]==viz[j])

{

ok=0;

break;

}

if(ok)

fout<<"DA";

else

fout<<"NU";

return 0;

}

# Cerinţa: Gengraf

Se dă un număr natural *n*. Construiți toate grafurile neorientate cu *n* vârfuri.

#include <fstream>

using namespace std;

ifstream fin("gengraf.in");

ofstream fout ("gengraf.out");

int a[101][101],n,x[16],e;

void scrie()

{

int p,i,j;

p=1;

for(i=1; i<=n-1; i++)

for(j=i+1; j<=n; j++)

{

a[i][j]=a[j][i]=x[p];

p++;

}

for(i=1; i<=n; i++)

{

for(j=1; j<=n; j++)

fout<<a[i][j]<<" ";

fout<<endl;

}

fout<<endl;

}

void gen(int k)

{

int i;

if(k==e+1)

scrie();

else

for(i=0; i<=1; i++)

{

x[k]=i;

gen(k+1);

}

}

int main()

{

int i,p;

fin>>n;

e=n\*(n-1)/2;

p=1;

for(i=1; i<=e; i++)

p=p\*2;

fout<<p<<endl;

gen(1);

return 0;

}

# Cerinţa: Graf Complet

Se dau mai multe grafuri neorientate, prin matricea de adiacență. Să se verifice despre fiecare graf dacă este complet.

#include <fstream>

using namespace std;

ifstream fin("graf\_complet.in");

ofstream fout ("graf\_complet.out");

int a[101][101],n;

int complet(int n)

{

int i,j;

for(i=1; i<=n-1; i++)

for(j=i+1; j<=n; j++)

if(a[i][j]==0)

return 0;

return 1;

}

int main()

{

int i,j,p,k;

fin>>k;

for(i=1; i<=k; i++)

{

fin>>n;

for(j=1; j<=n; j++)

for(p=1; p<=n; p++)

fin>>a[j][p];

if(complet(n))

fout<<"DA";

else

fout<<"NU";

fout<<endl;

}

return 0;

}

# Cerinţa: Subgraf

Se dă lista muchiilor unui graf neorientat cu *n* vârfuri, etichetate de la *1* la *n*. Din acest graf se elimină toate vârfurile etichetate cu valori prime. Să se determine câte muchii va avea subgraful obținut.

#include <fstream>

using namespace std;

ifstream fin("subgraf.in");

ofstream fout ("subgraf.out");

int a[101][101],n,v[101];

int prim(int x)

{

int i;

if(x<=1)

return 0;

for(i=2; i\*i<=x; i++)

if(x%i==0)

return 0;

return 1;

}

int main()

{

int i,j,x,y,m=0;

fin>>n;

while(fin>>x>>y)

a[x][y]=a[y][x]=1;

for(i=1; i<=n-1; i++)

for(j=i+1; j<=n; j++)

if(a[i][j] && !prim(i) && !prim(j))

m++;

fout<<m;

return 0;

}

# Cerinţa: Grade

Se dă lista muchiilor unui graf neorientat. Să se afișeze gradul fiecărui vârf.

#include <fstream>

using namespace std;

ifstream fin("grade.in");

ofstream fout ("grade.out");

int a[101][101],n,v[101];

int main()

{

int i,j,x,y;

fin>>n;

while(fin>>x>>y)

a[x][y]=a[y][x]=1;

for(i=1; i<=n-1; i++)

for(j=i+1; j<=n; j++)

if(a[i][j])

v[i]++,v[j]++;

for(i=1; i<=n; i++)

fout<<v[i]<<" ";

return 0;

}

# Cerinţa: Lista vecini

Se dă lista muchiilor unui graf neorientat. Să se afișeze, pentru fiecare vârf al grafului, lista vecinilor săi.

#include <fstream>

using namespace std;

ifstream fin("listavecini.in");

ofstream fout ("listavecini.out");

int a[101][101],n,v[101];

int main()

{

int i,j,x,y;

fin>>n;

while(fin>>x>>y)

a[x][y]=a[y][x]=1;

for(i=1; i<=n-1; i++)

for(j=i+1; j<=n; j++)

{

v[i]+=a[i][j];

v[j]+=a[i][j];

}

for(i=1; i<=n; i++)

{

fout<<v[i];

for(j=1; j<=n; j++)

if(a[i][j])

fout<<" "<<j;

fout<<endl;

}

return 0;

}

# Cerinţa: Adiacenta1

Se dă lista muchiilor unui graf neorientat. Să se afișeze matricea de adiacență a grafului.

#include <fstream>

using namespace std;

ifstream fin("adiacenta1.in");

ofstream fout ("adiacenta1.out");

int a[101][101],n,m;

int main()

{

int i,j,x,y;

while(fin>>x>>y)

{

a[x][y]=a[y][x]=1;

if(x>n)

n=x;

if(y>n)

n=y;

}

for(i=1; i<=n; i++)

{

for(j=1; j<=n; j++)

fout<<a[i][j]<<" ";

fout<<endl;

}

return 0;

}

# Cerinţa: Adiacenta

Se dă lista muchiilor unui graf neorientat. Să se afișeze matricea de adiacență a grafului.

#include <fstream>

using namespace std;

ifstream fin("adiacenta.in");

ofstream fout ("adiacenta.out");

int a[101][101],n,m;

int main()

{

int i,j,x,y;

fin>>n>>m;

for(i=1; i<=m; i++)

{

fin>>x>>y;

a[x][y]=a[y][x]=1;

}

for(i=1; i<=n; i++)

{

for(j=1; j<=n; j++)

fout<<a[i][j]<<" ";

fout<<endl;

}

return 0;

}